OpenSESAME
The Simple but Extensive, Structured Availability Modeling Environment

Max Walter[a], Markus Siegle[b], and Arndt Bode[a]

[a]*Institut für Informatik 10, Technische Universität München, D-80290 München, Germany*

[b]*Institut für Technische Informatik, Universität der Bundeswehr München, D-85577 Neubiberg, Germany*

Corresponding author:

Max Walter, Institut für Informatik 10, Technische Universität München, D-80290 München, Germany

Email: walterm@in.tum.de Tel: +4989 289 18456 Fax: +4989 289 17662

Abstract

This article describes the novel stochastic modeling tool OpenSESAME which allows for a quantitative evaluation of fault-tolerant High-Availability systems. The input models are traditional reliability block diagrams (RBD) which can be enriched with inter-component dependencies like failure propagation, failures with a common cause, different redundancy types, and non-dedicated repair. OpenSESAME offers a novel set of graphical diagrams to specify these dependencies. Due to the dependencies, traditional solution methods for RBDs cannot be applied to OpenSESAME models. We therefore present a novel evaluation method, which is based on the automatic generation of several state-based models, which are semantically equivalent to the high-level input model. Alternatively, either stochastic Petri nets or textual models based on a stochastic process algebra can be generated. The state based models are then analyzed using existing solvers for these types of models. Three case studies exemplify the modeling power and usability of OpenSESAME.

Key words: dependability modeling, reliability block diagrams, stochastic dependencies, web server, adjunct processor, fault-tolerant water supply

1 Introduction

A quantitative analysis of fault-tolerant systems may serve several purposes: First, it is important to know the risk of using/operating the system. For instance, it is frequently quoted that components used in an airplane should have an unreliability of less than 10^{-9} for one hour of flight. Second, it is often not trivial how a fault tolerant system must be built in order to achieve the highest level of dependability, given a fixed set of components. A similar question is: which part of the system should be made more dependable to get the most increase in dependability? Third, increasing the dependability of a system by adding redundancy usually implies some performance losses and/or increased costs. Thus, the decreased performance/cost ratio must be justified by a credible increase in reliability or availability. Fourth, from a somewhat different perspective, a quantitative evaluation of the dependability of a system may help to make the right decisions when purchasing such a system.

Stochastic dependability models [1–5] are used to predict the dependability of systems from the dependability of their components. *Fault trees* (FT, see [6–8]) and *reliability block diagrams* (RBD) are the most common modeling techniques for fault tolerant systems. Unfortunately, the traditional solution methods for these models rely on the assumption that there are no inter-component dependencies between the components of the system. Therefore, these techniques yield dangerous, over-optimistic results as they do not take into account failures with a common cause, failure propagation, imperfect failure detectors, fail-over times (spent for fault detection, localization, and isolation as well as reconfiguration), physical disturbances (e.g. heat, electro-magnetic noise, vibration) and limited repair personnel.

State based models like Markov chains can be used to model systems including these dependencies. As the state space grows exponentially with the number of components, it is usually implicitly defined using e.g. stochastic process algebras [9,10] or stochastic Petri nets [11–15]. However, these formal and rigorous modeling methods are, in comparison to e.g. fault trees, not very intuitive and require a higher learning effort. Furthermore, it is difficult to modularize these models and arrange them in a hierarchic manner which often results in quite unreadable models. Moreover, a stepwise refinement of the models is usually not possible.

On this account, we propose to not use state-based models for specifying the system directly, but to automatically generate them from a high-level model description. In this paper, we advocate the tool OpenSESAME (Simple but Extensive Structured Availability Modeling Environment, [2,16–18]) for this purpose. The tool offers a user-friendly modeling environment which is based on well-known formalisms like reliability block diagrams. However, in contrast to conventional tools, these diagrams can be enriched by inter-component dependencies including different kinds of redundancy strategies (like active-active or active-standby), failure propagation, failures with a common cause, limited repair and so on. Sec. 2 describes the input model of OpenSESAME in more detail. Sec. 3 explains how the input model is transformed into a state-based model, which can then be evaluated by existing tools. For the modelers, the transformation process is invisible: they just tell OpenSESAME to perform the calculations and obtain the results. Thus, no knowledge in the area of Petri nets or stochastic process algebra is necessary to create and analyze OpenSESAME models. This is exemplified in Sec. 4 which presents three real-world examples. The related work is described in Sec. 5. Finally, Sec. 6 summarizes the paper and gives an outlook on open problems and future work.

2 Input Model

An OpenSESAME model as seen by the user comprises component tables, reliability block diagrams, failure dependency diagrams, repair group tables, and variable tables. Not all model parts are necessary, usually one starts with one component table and a block diagram, only. Then, the model can be refined by adding additional tables and diagrams. In the following, the individual parts of a model are described.

2.1 Components

The *component tables* list all components the system consists of. Each component type has a unique name, a mean time to failure (MTTF), and a mean time to repair (MTTR). In its current implementation, OpenSESAME assumes exponentially distributed failure and repair times, only. If the system contains several components of the same type, the table also lists the number of components of this type. Furthermore, each component may have either a dedicated repair person or is allotted to a repair group (see below). For small and medium sized models, a single component table will be sufficient. For larger models, several tables can be used to group related components. As a rule of thumb, each table should not contain more than about ten different components to conserve readability.

2.2 Reliability Block Diagrams

An extended version of *reliability block diagrams* (RBD) is used to model the redundancy structure of the system. Defining the redundancy structure implies the specification of the measures which will be computed by OpenSESAME. RBDs are undirected graphs where each edge is labeled with a component. A component may appear several times in the same RBD. Several pairs of terminal nodes s and t each define a Boolean system which is available, if there is a connection between these nodes and unavailable otherwise. As components can be unavailable so can the edges: calculating the probability whether s and t are connected yields the availability of the modeled system. During a solution process, OpenSESAME calculates this availability. A special variable t (see 2.6) is used to specify the instant of time for a transient analysis. If t is set to ∞, a steady-state analysis is performed.

In OpenSESAME, several modern extensions to traditional RBDs were implemented. First, so-called k-out-of-N:G edges are supported to allow for a compact representation for many real-world systems. As its name implies, a k-out-of-N:G edge is available, as long as at least k out of its N parallel paths are available (or *good*). The parameter k is usually depicted in a circle right to the k-out-of-N:G edge. Fig. 1 shows three representations of a 2-out-of-3:G system comprising the components c[1], c[2], and c[3] which were previously defined in the component table. The leftmost representation refers to an array of identical components in the table and is the most compact one. Its size is independent of the parameters N and k, whereas the rightmost representation may become very large even for small values of N and k ($\binom{N}{k}$ paths each consisting of k edges are needed).

Second, the user may specify several pairs of terminal nodes. This allows for calculating the availabilities of subsystems in addition to the overall availability.

Third, edges may be labeled with a sub-RBD instead of a component. This allows for building a hierarchy of RBDs. Thus, even large systems can be modeled in a concise way. For example, Fig. 2 shows a hierarchic set of RBDs. The topmost diagram “main” is the top-level RBD describing a parallel system. Each path of the system is described in more detail in a sub-diagram. For the sake of simplicity, the figure shows only one of these paths, namely path 3. Path 3 is a series system: both valve A and valve B must be available for this path, as well as at least k out of N pumps. Each pump is again modeled by a sub-diagram. Because all sub-diagrams are identical, an array of sub-diagrams (pump[i]) is used. Each pump is again a series system comprising two components called part1 and part2. The basic edges valve A, valve B, part1[i], part2[i] refer to the respective components in the component table (Tab. 1).

2.3 Redundancy Types

As a unique feature of OpenSESAME, the model can be enriched with inter-component dependencies. Because some dependencies are related to the redundancy structure of the system, it makes sense to add these dependencies to the RBD. For example, in many systems fault tolerance is achieved using passive or standby redundant components. In such systems, the failure of an active component must be detected, localized and isolated, and a system reconfiguration is necessary after the failure of the primary component. During this so-called fail-over time, the system is unavailable (see Fig. 3).

To avoid over-optimistic results and unfair comparisons, availability models should therefore take into account possible fail-over times. In OpenSESAME, k-out-of-N:G edges can therefore be attributed with a fail-over time. Please refer to Sec. 4.2 of this article for more detailed information.

2.4 Failure Dependency Diagrams

Not all inter-component dependencies result from fault tolerance mechanisms. They can therefore appear between any two or more components of the system, even if they are not neighbors in the redundancy structure. Adding these dependencies to the RBD might therefore be difficult. For instance, it might happen that there is a failure propagation between two components which were modeled in different sub-RBDs.

To explicitly make a distinction between the redundancy structure on the one hand, and failure dependencies on the other hand, OpenSESAME supports a second kind of diagram, called *failure dependency diagram* (FDD). These diagrams can be used to model failures with a common cause, different kinds of failure propagation as well as imperfect failure detection mechanisms (imperfect coverage).

FDDs are directed graphs: the nodes are labeled with components, whereas the edges are labeled with a type and a probability. Currently, two types of failure dependencies are supported: destructive failure propagation and blocking failure propagation. Drawing an edge of destructive failure propagation with probability p between two components S and T means, that whenever S fails, the failure is propagated to T with probability p. Thus, T fails at the same time and must be repaired, too. Destructive failure propagations are drawn using solid lines whereas blocking failure propagations are drawn by dashed lines (see Fig. 4a). In the blocking case, T becomes unavailable but will not be repaired by itself. As soon as component S is repaired, component T becomes available, too.

As an FDD can contain several edges, and one OpenSESAME project can comprise several FDDs, sophisticated failure dependencies can be modeled in a concise and clear way. For example, Fig. 4b) and 4c) show several possible dependencies originating at source component S and affecting two target components T_1 and T_2. In Fig. b) the targets are independently affected: the failure may either propagate to T_1, T_2, or both T_1 and T_2. The probability for the latter is – as the term independent implies – p^2. In contrast, Fig. c) shows common failure propagation: here, either both components T_1 and T_2 are affected by the propagation, or none. This behavior is modeled by using a so-called pseudo component P. P does not appear in the redundancy structure and cannot fail (its MTTF is ∞). However, it might be the target of a failure propagation from component S. In that case, it propagates the failure to both T_1 and T_2 with probability 1.

For another example how FDDs can be used, please refer to Fig. 4d). Here, the pseudo component P is the source of a failure propagation. This FDD can be used to model common cause failures. P is attributed with the mean time between such events. If P fails, i.e. the common cause occurs, a failure is induced in both T_1 and T_2.

In standby redundant systems, a failure of the active component has to be detected before the system can be reconfigured. This does not only take some time (see Sec. 2.3), but may not be successful at all (no failure detection mechanism is perfect). Imperfect failure detectors can be modeled with OpenSESAME using failure propagation diagrams as shown Fig. 5 (left). A pseudo component is used as a target of a non-destructive failure propagation. The probability of propagation is $1 - c$, where c is the coverage factor of the failure detection mechanism. This time, the pseudo component also appears in the redundancy structure of the systems, as shown in Fig. 5 (right).

Because a component may appear several times in an FDD, the diagrams can be easily modularized. In addition, an FDD node may refer to an array of similar components to allow for a compact representation of symmetric models. Please refer to Fig. 6 for an example.

2.5 Repair

Stochastic dependencies may also occur due to a limited amount of repair personnel. Unless every component has a dedicated repair person, delays in the repair may occur if multiple components are failed at the same time and the repair personnel is overburdened. In OpenSESAME, this can be modeled by means of a *repair group table*. Each entry of this table describes a repair group in terms of its size, i.e. number of repairs which can be done concurrently. In its current implementation, OpenSESAME assumes a non-interruptive, random-next repair policy for each repair group. Later revisions of OpenSESAME might also support different repair strategies like first come (i.e. failed) first served, shortest repair first or closest repair next etc.

2.6 Variables

In many scenarios, a model must be analyzed with a varying set of parameters. For instance, one might want to evaluate the availability of a system with varying failure and repair rates or a varying number of redundant components. It is a big advantage of OpenSESAME, that *all* parameters of the model can be defined in the form of a variable.

For this purpose, OpenSESAME supports three types of variables:

- *Mean times* represent positive real numbers and are used for MTTF and MTTR values, as well as fail-over times in RBDs.
- *Probabilities* are numbers from the interval [0:1] and are used to attribute edges in FDDs.
- *Naturals* can be used to specify the number of components of one type, the parameter K in k-out-of-N:G edges and the capacity of a repair group.

For example, Fig. 7 shows a generic model of a system comprising N components. The system is available, if at least K components are available. If one of the components fails, with probability p, all other components become blocked until this component is repaired. During the evaluation process, the values for N, K, p and the MTTF and MTTR of the component can be automatically substituted. This allows for an easy execution of experiment series and reproducible results.

There is one special time variable t which is used to specify the instant of time for a transient analysis. For a steady state analysis, t can be set to ∞.

3 Transformation Process

As OpenSESAME models contain inter-component dependencies between the failure and repair events, they cannot be evaluated with classical evaluation techniques for Boolean models. Thus, the input diagrams are transformed into state-based models, which in turn can be analyzed using an evaluation tool for this model class. Currently, Generalized Stochastic Petri Nets (GSPN, [13,19]), or, alternatively, textual models defined by a Stochastic Process Algebra, (SPA) [9] are created.

At a glance, the transformation process consists of nine phases as shown in figures 8 and 9. In the following, each phase is described in more detail.

3.1 Pre-processing

In the first phase, the model is simplified to facilitate the remaining transformation process. More specifically, all variables are substituted by the respective constants (if several experiments were defined, this and all remaining steps are repeated for each experiment). Then, component and RBD arrays are replaced by the respective number of normal components and RBDs. Finally, all hierarchies are resolved by copying the elements of the child diagrams into their parents. Likewise, all FDDs are merged (by copying the nodes and edges) into a single FDD. The result of the pre-processing phase is one OpenSESAME project for each experiment containing: one component table, one RBD, zero or one repair group table, and zero or one failure dependency diagram.

3.2 Gain redundancy structure

Next, the resulting RBD is converted into a symbolic expression. This can be done by finding all minimal paths of the RBD. Each path then represents one conjunction-term of the redundancy structure given in disjunctive normal form. Thus, the redundancy structure is of the form:

$$\bigvee_{i=1,2,\ldots,n} (c_{i1} \wedge c_{i1} \wedge \ldots \wedge c_{im})$$

In the worst case, the number of paths grows exponentially with the size of the RBD. Thus, we are currently implementing a method to directly convert the RBD into a binary decision diagram based on methods which are known to work well for a diverse set of graphs [20].

3.3 Find independent sub-models

With the required information at hand we could now create one big state-based model for the overall OpenSESAME project. However, as the number of states in such a model grows exponentially with the number of components, the project is divided into parts which can be solved individually by finding disjoint *sets of inter-dependent components (SIC)*. This is done by creating an undirected graph whose nodes are the components of the system. If two components are dependent, an edge is added between the respective nodes. The SICs are generated by finding the maximal connected sub-graphs, e.g. by a depth first search.

For example, the system modeled in Fig. 10 comprises the four components $\{A, B, C, D\}$. Due to the FDD, there are stochastic dependencies between A and B, as well as between C and D. However, e.g. A and C are independent from each other. Thus there are two sets of inter-dependent components: $\{A, B\}$ and $\{C, D\}$.

3.4 Decompose redundancy structure

The divide and conquer approach described above also requires to divide the redundancy structure according to its variables. A simple way to do this is by using a method similar to the Shannon decomposition which is best shown by referring again to the example shown in Fig. 10. Its redundancy structure ϕ looks like this:

$$\phi = (A \wedge B \wedge C) \vee (A \wedge B \wedge D) \vee (A \wedge C \wedge D) \vee (B \wedge C \wedge D)$$

As the system consists of two SICs, two state-spaced models will be created. Defining, e.g. $\phi_{A\overline{B}}$ as ϕ with A substituted by `true` and B substituted with `false`, i.e.:

$$\begin{array}{ll} \phi_{AB} = (C \vee D) & \phi_{A\overline{B}} = (C \wedge D) \\ \phi_{\overline{A}B} = (C \wedge D) & \phi_{\overline{AB}} = \texttt{false}, \end{array} \tag{1}$$

the redundancy structure ϕ is divided as follows:

$$\begin{aligned} \phi = & [(A \wedge B) \wedge \phi_{AB}] \vee [(A \wedge \overline{B}) \wedge \phi_{A\overline{B}}] \vee \\ & \vee [(\overline{A} \wedge B) \wedge \phi_{\overline{A}B}] \vee [(\overline{A} \wedge \overline{B}) \wedge \phi_{\overline{AB}}] = \\ = & [(A \wedge B) \wedge (C \vee D)] \vee [(A \wedge \overline{B}) \wedge (C \wedge D)] \vee \\ & \vee [(\overline{A} \wedge B) \wedge (C \wedge D)] \vee [(\overline{A} \wedge \overline{B}) \wedge (\texttt{false})] \end{aligned}$$

As all sub-terms are disjoint, $Pr\{\phi\}$, i.e. the probability that ϕ is true, can be computed by:

$$\begin{aligned} Pr\{\phi\} = & Pr\{(A \wedge B) \wedge (C \vee D)\} + Pr\{(A \wedge \overline{B}) \wedge (C \wedge D)\} + \\ & + Pr\{(\overline{A} \wedge B) \wedge (C \wedge D)\} + 0 \end{aligned}$$

and finally, because the subsets $\{A, B\}$ and $\{C, D\}$ are stochastically independent, it holds:

$$\begin{aligned} Pr\{\phi\} = & Pr\{(A \wedge B)\} \cdot Pr\{(C \vee D)\} + \\ & + Pr\{(A \wedge \overline{B})\} \cdot Pr\{(C \wedge D)\} + \\ & + Pr\{(\overline{A} \wedge B)\} \cdot Pr\{(C \wedge D)\} \end{aligned}$$

In the following, a state state-based model is created for the SIC $\{A, B\}$, which is used to compute the probabilities $Pr\{A \wedge B\}$, $Pr\{(A \wedge \overline{B})\}$, and $Pr\{(\overline{A} \wedge B)\}$ and a model is created for the SIC $\{C, D\}$, to compute the remaining probabilities. This is done by attributing the state space models by respective reward rates. For example, the reward rates of the model for SIC $\{A, B\}$ are: $(A \wedge B)$, $(A \wedge \overline{B})$, and $(\overline{A} \wedge B)$. In the solution process, the solver will compute the probabilities of these reward rates which can then be used to compute $Pr\{\phi\}$.

3.5 Create component oriented data structures

The OpenSESAME diagrams contain information in a dependency-centric view. For instance, the FDDs contain information on the failure dependencies using directed edges. This allows for an easy adding, removing and modifying of dependencies. However, to generate a state-based model a component-centric system view is required.

Thus, as a fifth step, each SIC is converted into a component-oriented data structure. This data structure comprises one object for each component of the SIC, which is attributed by the inter-dependencies this component is involved in. For example, Fig. 11 shows an extract of a UML object diagram of such a data structure. The component shown is a member of a redundancy cluster, and can be affected by a destructive failure-propagation from other components. In turn, if the component fails, with a probability of 0.001, the failure is propagated to other components. The component is repaired by a single, non-dedicated repairman.

3.6 Create Petri net / algebraic model

From this component-centric model, a state-based model can be derived. Currently, two basic types of state-based models are supported by OpenSESAME: Generalized Stochastic Petri Nets (GSPNs, [13]) and textual models based on the stochastic process algebra supported by the tool CASPA [10]. In the following, we will describe how Petri nets are created, a similar approach was chosen for the algebraic models.

One Petri net is created for each SIC. Each Petri net consists of several subnets: one for each component of the SIC, one for each redundancy group, and one for each repair group. In a second step, these subnets are connected to create the overall Petri net. For an introduction to stochastic Petri nets, please refer to [13] or [12]. In this article, we follow the suggestions found in [1] to obtain a concise graphical representation of the nets.

The structure of a component subnet depends on the inter-component-dependencies the component is involved in. For example, Fig. 12 shows the Petri net which is created for the component from Fig. 11.

Three places represent the main states the component may be in: stand-by, active and failed. Whether the component is in active or stand-by mode is determined by the state of other components. Thus, the transitions *T1* and *T2* will later be attributed by so-called *guards*, which are marking dependent transition enablings.

The transitions *T3* and *T4* represent component failures. If a failure is propagated to the component, this is done by moving a token into the place *P4*, also called a failure propagation input place (*FPIP*). This will immediately bring the component in the failed state by enabling the immediate transition *T7*.

In turn, if a failure occurs, with probability p, *T8* fires and an additional token will be created and moved to place *P5*, called the failure propagation output place (*FPOP*). From there, the token will be moved to *FPIPs* of other components. With probability $(1-p)$ *T9* fires instead of *T8* and no additional token is created.

In both cases, the component is repaired by a non-dedicated repair person. Thus, the repair may be delayed if the repair person is engaged in repairing some other components. This is modeled by the immediate transition *T5*. The repair itself is modeled by the timed transition *T6* which is attributed by the mean time to repair of the component.

After creating such a subnet for each component, these subnets are connected by appropriate edges. This includes connecting the *FPOPs* to their corresponding *FPIPs*, and the places representing repair groups to the corresponding transitions of the components which are repaired by these groups. In stand-by redundant systems, the transitions between the places *active* and *stand-by* have to be attributed by appropriate guards to ensure that at most k configurations can be in the active state.

3.7 Generation and analysis of Markov chain

For computing its state probabilities, each SIC has to be converted from its high-level description (Petri net or process algebraic specification) to the underlying continuous-time Markov chain (CTMC). Currently, input for the Petri net tools DSPNexpress [12] or Möbius [15], or for the stochastic process algebra tool CASPA [10] can be generated. During the generation of the CTMC, the notorious state space explosion problem may occur, since every possible interleaving of concurrent activities taking place in the individual components must be considered.

The tool DSPNexpress transforms the GSPN model into a CTMC by exploring the reachability graph of the Petri net in a conventional state-by-state fashion, the generation time thus being linear in the total number of MC-transitions found. Furthermore, the CTMC is stored explicitly in a sparse matrix representation whose memory requirement is linear in the number of transitions of the Markov chain. Therefore, as a rule of thumb, using DSPNexpress for a SIC containing more than, say, 20 components, the generated CTMC will not fit into main memory and consequently a solution will not be possible within reasonable time. For the computation of the stationary probability vector, DSPNexpress offers GMRES [21], a Krylov subspace method which is known to converge rapidly.

In order to manage models with very large state space, the tools Möbius (in connection with the pZDD engine described in [22,23]) and CASPA do not employ an explicit representation of the CTMC, but a symbolic representation based on decision diagram data structures. In the following, the strategy implemented in the Möbius pZDD engine is briefly sketched, for details of CASPA the reader is referred to [10]. The Möbius pZDD engine uses a new type of data structure, called partially shared zero-suppressed multi-terminal BDD (pZDD for short), which extends zero-suppressed BDDs [24] to the multi-terminal case and guarantees maximal sharing of subgraphs (and therefore increased compactness). For generating the Markov chain from the GSPN, the activity-local approach [22] performs only a partial explicit exploration of the state graph, which is combined with a symbolic encoding and subsequent symbolic composition of activity-specific substructures. Since the composition step generates a potential state space, i.e. a superset of the actually reachable transitions, the approach is complemented by an efficient symbolic reachability algorithm (based on sets of states instead of individual states). Numerous experiments have shown that such a scheme creates very memory-efficient symbolic representations of even huge CTMCs (theoretical results characterize a setting where the size of the decision diagram grows only linearly in the number of concurrent components [25]). The key to such compactness lies in some heuristics for transition encoding which exploit the inherent regularity of the CTMC. Furthermore, the method is also very runtime efficient, e.g. for a 6-out-of-8 version of the Adjunct Processor described in Sec. 4.2 a CTMC with more than 261 million states and 5.8 billion transitions is generated in less than 20 seconds on an AMD Opteron 2.4 GHz computer system [26]. The Möbius pZDD engine performs numerical analysis of the CTMC, reading the transition rates from the symbolic representation of its transition matrix while using an explicit representation (i.e. ordinary arrays) for the iteration vectors. Such a combination makes the numerical analysis of models with hundreds of millions of states possible. The symbolic engine offers the Jacobi iteration scheme, as well as a block-oriented so-called pseudo Gauss-Seidel iteration scheme [27]. The numerical analysis of the above 261 million state CTMC requires 4.1 GByte of RAM (95% of which is consumed by the iteration vectors, not by the CTMC!) and can be accomplished on the same Opteron system in about 30 minutes. Clearly, a CTMC of this size can neither be generated nor solved within reasonable time if one uses only explicit data structures.

Similar to changing the back-end of a compiler in order to migrate it to a new architecture, tools such as GreatSPN [13], SHARPE [19], TimeNet [11] or WebSPN [28] could be used as back-end evaluation tools of OpenSESAME, thereby exploiting their specific features (e.g. the analysis of non-Markovian models). In our ongoing work, we are investigating these tools and their applicability to OpenSESAME models.

3.8 Compose results & Post-processing

Whatever solution-method is used, the result will be a set of probabilities for each SIC, which have to be combined to a single result according to the rules shown in Sec. 3.4. Additionally, other post-processing, like summarizing the results of several solution runs with different parameter assignments in a table or diagram (see e.g. Fig. 22), can follow.

4 Case Studies

The following sections contain three realistic examples showing how Open-SESAME can be used to easily model fault-tolerant systems including inter-component dependencies. In the first example, a fault-tolerant web server, OpenSESAME is used to determine the appropriate number of redundant components. The second example stems from an industrial cooperation and models a real-world telecommunication system. The model is made up of several RBDs which are arranged in a hierarchic manner and shows how standby redundant systems are modeled. The last example addresses a water-supply system of a city requiring a more complex reliability block diagram and common failure propagation.

4.1 Web server

Fig. 13 shows the structure of a typical fault-tolerant web server. Usually, *HTTP servers* act as a front-end and receive requests from clients. So called *application servers* are responsible for the creation of dynamic web pages. They receive requests from the HTTP servers and provide the respective HTML files. The *database servers* are used to store the raw data which is used to create static and dynamic web pages. Neither the HTTP-servers nor the application servers can operate without access to the database servers.

We further assume that there is more than one computer of each type to allow for the necessary bandwidth and an acceptable response time for all clients. We assume that the minimum number of necessary servers are N_H, N_A, and N_D, respectively.

Additionally, the architecture contains a number of spare servers of each type for the sake of increased dependability. It is one of our goals of the following availability analysis to determine the necessary number of redundant servers. At the moment, these numbers are called R_H, R_A, and R_D, respectively.

Last but not least, there is an interconnection network between the servers. A network failure will lead to an unavailability of the web server architecture.

Tab. 2 lists the components of the web architecture and their respective MTTF and MTTR values. For the servers we assume an average uptime of one week (168 hours) as this is the mean uptime for a standard PC with a standard operating system [29]. As most failures are due to software bugs, the *repair* of a PC corresponds to rebooting the system which is assumed to take no longer than 30 minutes.

The network, which does not comprise any sophisticated software mechanisms, has an MTTF of a million hours (i.e. 114 years). This value is quoted by vendors of this type of equipment [30]. For the repair of the network we assume that 24 hours are needed on the average.

The redundancy structure of the web server is specified by the RBD shown in Fig. 14. It is a series system comprising the subsystems HTTP-servers, application servers, database servers, and network. Each of the subsystems (except the network) is a N-out-of-(N+R):G system where the number (N+R) is the total number of servers (including redundant servers) and N is the number of servers which are necessary to guarantee for the specified minimum performance. The network is assumed to be non-redundant and represents a so called single point of failure.

Tab. 3 shows the results computed with the tool OpenSESAME under the assumption that $N_H = N_A = N_D = 2$ and for different values for $R_{H,A,D}$. The table shows also the respective unavailability $U = 1 - A$ and the mean annual costs due to down-time under the assumption that one hour of down-time costs 8000 €. As can be seen at least one redundant server must be used for each server type to avoid intolerable down-time costs. Moreover, using more than two redundant servers of each type is clearly not justified as the increase in availability is negligible.

Until now, we assumed that the failure and repair times of all components were statistically independent. However, in a real web server, there will be inter-dependencies between its components. For example, failed nodes may jam the network by sending illegal packages. As the distributed web server has no redundant network, such a *babbling node* brings the system down.

Therefore, a meaningful availability analysis should take the phenomenon of babbling nodes into account. With OpenSESAME, this can be done by attributing the model with a non-destructive failure-propagation. In detail, we assume that every time a server fails there will be a failure propagation to the network with probability p.

Fig. 15 shows the FDD modeling the phenomenon of babbling nodes. An evaluation of the OpenSESAME model including this FDD yields the results which are summarized in Tab. 4. As can be seen, the unavailability increases with the probability of failure propagation. If p is large (i.e. $p \geq 0.01$), adding redundant servers will not contribute to a higher availability. In contrast, adding redundancy will *decrease* the system's availability because there are more possible sources for failures. Thus, the only solution is to either reduce the risk of babbling nodes, e.g. by using switched networks, or adding a redundant network to tolerate the babbling nodes.

Because there is a dependency between the LAN and every node, the model cannot be divided into several SICs prior to its solution. In contrast, OpenSESAME creates a single GSPN to evaluate the model.

4.2 Adjunct Processor

A traditional application area for highly available systems is telecommunication. Although a single call seems to be of little financial value, losing several thousands of calls due to a failure will lead to a substantial loss for the telecommunication provider.

One of the critical components in a modern digital telecommunication network is the so-called adjunct processor (AP). Its task is, among others, to translate virtual phone numbers into location dependent physical numbers. This allows for easy-to-remember and location independent phone numbers. E.g., in Germany, 110 can be called from every telephone to reach the police. However, the specific police department the callers will be connected to depends on their current location.

4.2.1 Components of the Adjunct Processor

Fig. 16 shows a schematic drawing of a digital adjunct processor implementation. It comprises several single board computers (SBC) which are plugged into a passive backplane. This backplane serves as a means for communicating data and control information between the computers. Each SBC is connected to a dedicated rear transition board (RTB) which is plugged into the other side of the backplane. All I/O-devices which would normally be connected to the SBC are connected to its RTB instead. This makes it possible to remove an SBC from the backplane without removing any cabling. Thus, a failed board can be replaced very quickly. Moreover, modern standardized backplanes like AdvancedTCA [31] allow for removing boards during runtime, what is usually referred to as the hot-swap capability.

The backplane houses two different kinds of SBCs: I/O-SBCs, which run a telecommunication protocol stack (e.g. SS7) and connect the adjunct processor to the rest of the world, as well as host-SBCs, which run a database and are responsible for the actual translation between virtual and physical phone numbers. To perform their tasks, the RTB of each I/O-SBC is connected to the telecommunication network via an E1/T1 line. The RTBs of the host-SBCs are connected to mirrored disks storing the mapping of phone numbers.

The detection and isolation of faulty boards, as well as the control over removing and inserting boards lies within the responsibility of a so-called hot-swap controller. In addition to all these electronic components the system also comprises several fans for cooling and power supplies. Tab. 5 summarizes these components and shows their respective MTTF and MTTR values.

The MTTF values are field data obtained by the suppliers of the products. Typically, to estimate the MTTF of a component which lacks the necessary amount of field data, standardized methods like the MIL-HDBK-217 [32] or the Siemens Norm SN29500 [33] can be used. The data listed here was partially computed via the Siemens Norm.

In the adjunct processor, repairing a component means replacing it by a new one. Under ideal conditions, we can assume the repair time of a single board computer to be 5 minutes, including the reboot process of the operating system on the board. For the repair of an RTB, cabling has to be removed from the failed board and attached to the new board after replacement. Thus, the overall repair process is assumed to take 30 minutes. After the failure and replacement of one of the RAID disks, data has to be copied to the newly installed disk, which is assumed to take 30 minutes, too.

The only component, which cannot be replaced that easily, is the system's backplane. In case of a backplane failure, the whole system has to be disassembled and put together with a new backplane, again. For this procedure, we assessed one hour of repair time.

4.2.2 Redundancy structure of the Adjunct Processor

For our example, a redundancy structure as shown in Fig. 17 is proposed for the Adjunct Processor. The model comprises five reliability block diagrams: a high-level diagram for the overall system and two secondary diagrams representing a host-unit and an I/O-unit respectively. The remaining sub-RBDs are used to specify that each unit comprises an SBC and the corresponding RTB.

A simple combinatorial analysis of this model shows that the unavailability of the adjunct processor is $2.00 \cdot 10^{-7}$ corresponding to an availability of 99.99998 %. Thus, the availability is in the order of nearly "seven Nines", where usually only "Five Nines" are required in the telecommunication environment. However, the model does not take into account any inter-component dependencies. Hence, it is likely that the model is over-optimistic and that the real availability of the adjunct processor will be lower than the value computed.

4.2.3 Inter-component dependencies in the adjunct processor

A typical dependency between redundant components which is neglected when using standard Boolean modeling methods is the fact that a standby redundant component has to be reconfigured before it can take over the task of a failed active component.

These fail-over times are known to contribute significantly to the down-time of any system in the telecommunication environment. In the adjunct-processor we have to assume that there is a fail-over time after the failure of a host-unit, as well as after the failure of an I/O-unit. The disks, fans, HSCs and power supplies are arranged in an active-active redundancy scheme and hence have no fail-over time.

For the host-units, failure detection can be done by using a mutual heartbeat-protocol with a period of one second. Thus, in the worst case, it will take a second for the standby-node to detect the failure of the active node. As the standby unit is in warm redundancy, it can replace the active unit immediately. Hence, the fail-over time between host-units is one second. A failure of an I/O-unit is assumed to be detected by the external telecommunication switch which unsuccessfully tries to reach the adjunct processor via the I/O-unit. After failure detection, the switch informs the HSC about the failed I/O-unit and reconfigures its internal table on available I/O-units. It is assumed that the whole process can be done in less than 3 minutes.

Fig. 18 shows how the non-zero fail-over times can be integrated into an OpenSESAME Redundancy Diagram. Compared to Fig. 17, the only differences are the dashed edges representing an active/standby redundancy scheme. The fail-over times themselves are depicted as a textual attribute above the respective edges.

An evaluation of this model with OpenSESAME yields an unavailability of $7.42 \cdot 10^{-7}$ for the adjunct processor. This implies that its unavailability and the mean annual down-time costs were underestimated by the factor 2.5 using the simple model from Sec. 4.2.2. Even with the more realistic model, the system fulfils the requirements of “Five Nines” availability, though.

4.2.4 Computational Costs and Memory Requirements

Although there is not much of a difference from the modeler's point of view between the simple and the more realistic model, the solution process significantly differs. In the simple case, there are no dependencies and a dedicated SIC is created for each component. The unavailability of each component can be computed by applying the formula $U = \frac{MTTR}{MTTR+MTTF}$; no Markov chain must be created nor analyzed.

For the more realistic models, one SIC for the I/O-units and one SIC for the host-units is created. For the remaining components, dedicated SICs are generated. The first SIC is the largest and by far the most computationally challenging one: it contains 6 I/O-boards and 6 RTB-boards. The generated state space consists of nearly one million states and 7.4 GByte of main memory are required to solve this SIC with the tool DSPNexpress.

It is therefore better to solve SICs with more than about 10 components by using symbolic methods like the Möbius pZDD engine. Only 36 MByte of memory are necessary to solve the model with this technique.

In our future work, we will decrease the sizes of the generated state-spaces by exploiting symmetries in the model. For example, in the adjunct processor, all I/O-boards have the same MTTF and MTTR values, and the fail-over times are identical for all I/O-units.

4.3 Water Supply System

For a final illustrative example, the reader is referred to Fig. 19, showing a water supply system of a city. It consists of up to five pumps (1, 2, ..., 5) and two distributors (A and B). The water of pumps 1 and 2 flows into A whereas the water of pumps 4 and 5 flows into B. The water of pump 3 can flow into both A or B. We assume that there is enough water for the city as long as it is supplied by water from at least two pumps flowing through A, B, or a combination of both.

We assume that the pumps and distributors have a mean time to failure of one year. All pumps and distributors fail independently from each other. Immediately after a failure, pumps and distributors are being repaired, which takes 24 hours in average.

Pump 3 is special in terms of its failure behavior: after a failure, with probability p, the pump delivers dirty water which spoils any distributor to which it is delivered. Consequently, the distributor must be shut down until the pump is repaired. This raises the question which distributor should be connected to pump 3. In the following, we will compare three possible configurations C1, C2, and C3:

- C1: To avoid any possible contamination, pump 3 is not used at all.
- C2: Pump 3 is connected to both distributors to achieve the highest degree of redundancy.
- C3: As a compromise, pump 3 is connected to distributor A only.

The left RBD of Fig. 20 models configuration C1. If both distributors are available, the middle path can be chosen. This path is available, if at least 2 of the 4 pumps are available. If distributor B is failed, the upper path must be taken. This path requires the pumps 1 and 2 to be available. Similarly the lower path requires the pumps 4 and 5 to be available and must be taken, if distributor A is failed. An evaluation of this model with OpenSESAME yields an unavailability of the city's water supply of $3.73 \cdot 10^{-5}$. On average, the city is without water for 19.6 minutes per year.

The block diagrams for C2 and C3 are also shown in Fig. 20. These models cannot be evaluated using traditional solution methods due to the failure propagation from pump 3 to the distributors. The corresponding OpenSESAME-model is enriched with the failure dependency diagram shown in Fig. 21. For C2, the parameter B takes the value 1: a failure of pump 3 will first propagate to the pseudo-component P with probability p, and from there to both distributors A and B. The pseudo-component P cannot fail by itself and cannot be repaired (its MTTF and MTTR are both infinite). It ensures that A and B fail simultaneously in case of a failure propagation. The failure propagation is of the blocking type: as soon as pump 3 is repaired, the pseudo component as well as both A and B become available immediately. For C3, the parameter B takes the value 0. Thus, a pump failure can affect distributor A only.

An evaluation of the configurations C2 and C3 using OpenSESAME yields the results shown in Fig. 22. It shows the unavailability of the three configurations with respect to the parameter p.

The unavailability of C1 is constant as it does not depend on p. For very small values of p, C2 yields the best unavailability (between 3.96 and 11.1 minutes per year). However, the unavailability increases rapidly when p becomes larger.

In this case, pump 3 acts as a single point of failure, as it can spoil the overall system. Thus, if p is larger than 1 % , C2 is the better configuration. Finally, for large values of p, i.e. $p > 0.7$, it is better to not use pump 3 at all which results in the above-mentioned unavailability of 19.6 minutes per year.

For this example, the computational demands are negligible, as the largest SIC contains three components, only.

5 Related Work

5.1 SHARPE

The Symbolic Hierarchical Automated Reliability and Performance Evaluator (SHARPE) is an environment which allows the combination of several different modeling formalisms like block diagrams, fault trees, Markov Chains, Stochastic Reward Nets etc. (see [19]). In this way, the modeler can choose a different model type for different parts of the system and can therefore choose an appropriate level of manageability and accuracy for each part of the model. However, the tool does not allow to incorporate inter-component dependencies between two components which are modeled in two different diagrams. Thus, the set of components has to be divided into independent subsets at the very beginning of the modeling process. This division cannot be changed later, which hampers a stepwise refinement and the modifiability of the model.

5.2 MEADEP

The Measurement - Based Dependability Analysis Tool (MEADEP) [34] follows a similar approach as SHARPE. Here, Reliability Block Diagrams and Markov Chains can be combined into one availability model which is evaluated bottom up, i.e. the evaluation results of the lower models are used as parameters for the other models until the top-level model has been resolved. However, MEADEP does not allow to integrate inter-component dependencies to span over several diagrams.

5.3 HIDE

In contrast to SHARPE and MEADEP, the High-level Integrated Design Environment for Dependability (HIDE, [35]) and its successors [36] are not based on combining several modeling techniques but on converting a high level input model into state-based models. In other terms, HIDE uses a similar principle as the work presented in this paper. However, there are significant differences in the type of input diagrams. Whereas in our work the goal was to create simple, intuitive and manageable diagrams for modelers which are familiar with traditional combinatorial modeling methods, the HIDE project uses UML diagrams for input. This has the advantage that availability estimations can be gained “for free”, if such UML diagrams already exist, e.g. if they were created during the design phase of the system. However, if a model has to be created from scratch, many modelers will feel more comfortable with modeling methods which resemble traditional modeling methods for highly available systems, like the widely used Reliability Block Diagrams. There are also differences in respect to the kind of supported dependencies. For instance, unlike OpenSESAME, HIDE does not support blocking failure propagation nor a limited repair capacity.

5.4 DIFtree

Another tool which transforms high-level input diagrams into state-spaced models is Dynamic Innovative Fault Tree (DIFtree) [37,38], which has been incorporated into the modeling environment Galileo [39]. The input diagrams are so called Dynamic Fault Trees, which extend traditional fault trees by a set of new gates that can handle different kinds of redundancy (e.g. cold, warm, and hot redundancy). However, all dependencies have to be modeled by using these gates. Therefore, a failure propagation between two components in two different branches of the tree cannot be introduced without a major redesign of the model. Furthermore, many modelers prefer Reliability Block Diagrams to fault trees when modeling High Availability systems, as they are more closely related to the system's schematic layout.

5.5 Boolean Logic Driven Markov Processes

In a recent article [40], an innovative approach for combining fault trees and Markov models is presented. Each leaf of the fault tree represents a component of the system which can be described in more detail by a Markov process. Switching between the states of the chain can be triggered by the failure or repair of other components, which allows for modeling inter-component dependencies. To simplify the task of the modeler, several predefined standard cases can be reused. These cases include warm standby redundancy, on demand failures, and components with an increasing failure rate (i.e. aging components). The applicability of this technique to real world problems has been shown in the context of modeling fault-tolerant systems.

Unlike OpenSESAME, the state-based models are still visible in the user interface in this approach, as Markov chains are used to specify the inter-component dependencies. Another difference lies in the target application area: whereas OpenSESAME focuses on High-Availability systems, this approach aims at safety-critical, electrical high-voltage apparatus.

5.6 Dynamic Reliability Block Diagrams, DRBD

In this approach [41–43] dynamic extensions to reliability block diagrams are introduced. These extensions can be used to model varied dependencies between components including several types of redundancy (hot, warm, standby) and failure propagation. Dependencies are specified by their type (order or strong), the action and reaction events (wake-up, reparation, sleep, and failure). In total, 24 different kinds of dependencies are defined. The work also addresses the problem of conflicting dependencies, i.e. the behavior of the system in the case of multiple dependencies occurring at the same time. In contrast to OpenSESAME, all dependencies are specified in the RBD itself instead of additional diagrams. This hampers the creation of hierarchic diagrams which is one of the main goals of the OpenSESAME design. One should also note that at the moment there is no software implementation of DRBD yet. Further comparison will be made as soon as the implementation of DRBD is completed.

6 CONCLUSIONS & FUTURE WORK

This article is a contribution to the area of modeling fault-tolerant High-Availability systems including inter-component dependencies. We advocate an approach which is based on state-based modeling methods. However, unlike traditional techniques, which require to define the model by using stochastic Petri nets or models based on a stochastic process algebra, an input with a very high level of abstraction is used in the approach described here.

Generating the state-based models “by hand” is work-intensive, cumbersome and error-prone. As an alternative, we advocate the tool OpenSESAME, which allows for a stepwise refinement of simple Boolean models. Using OpenSESAME, one starts with a plain model, to which inter-component dependencies can be added at any time. No expert knowledge in the fields of Markov chains, Petri nets or Stochastic Process Algebras is necessary to create OpenSESAME-models.

To demonstrate the benefits of our approach, we modeled several fault tolerant systems in two ways: a simple RBD model was used without any inter-component dependencies, and a more elaborate model was used including inter-component dependencies like failure propagation, common cause failures, standby-redundancy and so on. We show that the simple models generate over-optimistic results which makes it advisable not to neglect the dependencies. However, the examples also show that by using OpenSESAME, integrating the dependencies into the models is very easy and can be done by a step-wise refinement of the original model.

In its current implementation, OpenSESAME does not take advantage from symmetries in the model. For example, if a model contains several identical components, the same Petri net structure is created as if the components were different. This may yield Petri nets with an unnecessary large state-space. Our future work will therefore concentrate on exploiting these symmetries to improve the performance of our tool.

Furthermore, OpenSESAME is quite limited in the kind of result which is generated: only transient and steady state availabilities are computed. Future versions of the tool should also be able to compute reliabilities, MTTF and MTTR values, and derivatives from these measures. Moreover, a sensibility analysis should be possible.

The main application area of OpenSESAME is – as its name implies – High-Availability. Most safety critical systems cannot be analyzed, as they usually have more then one undesired event (usually, a distinction between "failed-safe" and "catastrophic failure" is made). We therefore plan to extend the OpenSESAME concept in a way that it can be used for systems with more than the two macro states "available" and "failed".

7 AVAILABILITY OF THE TOOL

OpenSESAME can be obtained from `http://OpenSESAME.in.tum.de/` and runs on Unix-like operating systems. The GUI is written in Java and should run on most platforms including MS-Windows.

8 Acknowledgements

This work was partly funded by the Bayrische Forschungsstiftung in the projects *HotSwap* and *Balance*. We would like to thank our industrial partner Motorola GmbH for discussing the real-world case studies and the anonymous reviewers for their excellent suggestions and constructive criticism.

References

[1] W. G. Schneeweiss, Reliability Modeling, LiLoLe Verlag, Hagen, 2001.

[2] M. Walter, W. Schneeweiss, The modeling world of Reliability/Safety Engineering, LiLoLe Verlag, 2005.

[3] B. Haverkort, I. Niemegeers, Performability modelling tools and techniques, Performance Evaluation 25 (1) (1996) 17–40.

[4] K. Trivedi, G. Ciardo, M.Malhotra, R. Sahner, Dependability and Performability Analysis, in: Lecture Notes in Computer Science 729, 1993, pp. 587–612.

[5] M. Marseguerra, E. Zio, Basics of the Monte Carlo Method with Application to System Reliability, LiLoLe Verlag, 2002.

[6] W. G. Schneeweiss, The Fault Tree Method, LiLoLe Verlag, Hagen, 1999.

[7] Relex Software, Managing Reliability across the Product Lifecycle (2005).
URL `http://www.relexsoftware.com/`

[8] Reliasoft Corporation, Reliability Software, Training, Consulting and Related Services (2005).
URL `http://www.reliasoft.com/`

[9] H. Hermanns, U. Herzog, J.-P. Katoen, Process algebra for performance evaluation, Theor. Comput. Sci. 274(1-2) (2002) 43–87.

[10] M. Kuntz, M. Siegle, E. Werner, CASPA - A Tool for Symbolic Performance and Dependability Evaluation, in: Proceedings of First European Performance Evaluation Workshop, Springer, LNCS 3236, 2004, pp. 293 – 307.

[11] R. German, C. Kelling, A. Zimmermann, G. Hommel, TimeNet: A toolkit for evaluating non-Markovian stochastic Petri nets, Performance Evaluation 24 (1995) 69–87.

[12] C. Lindemann, Performance Modelling with Deterministic and Stochastic Petri Nets, Wiley and Sons, 1998.

[13] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, G. Franceschinis, Modelling with Generalized Stochastic Petri Nets, Wiley and Sons, 1995.

[14] C. Hirel, B. Truffin, K. Trivedi, SPNP: Stochastic petri net package. version 6.0, in: Proc. 11th International Conference on TOOLS, 2000, pp. 354–357.

[15] G. Clark, T. Courtney, D. Daly, D. Deavours, S. Derisavi, J. M. Doyle, W. H. Sanders, P. Webster, The Möbius Modeling Tool, in: Proc. 9th International Workshop on Petri Nets and Performance Models, IEEE Computer Society Press, 2001, pp. 241–250.

[16] M. Walter, C. Trinitis, How to Integrate Inter-Component Dependencies into Combinatorial Availability Models, in: Proc. Ann. Reliability and Maintainability Symp. (RAMS 2004), Los Angeles, USA, IEEE Computer Society Press, 2004, pp. 226–231.

[17] M. Walter, C. Trinitis, Simple Models for High-Availability Systems with Dependent Components, in: Proc. of the 2006 European Safety and Reliability Conference, Vol. 3, Taylor and Francis, 2006, pp. 1719–1726.

[18] M. Walter, C. Trinitis, OpenSESAME: Simple but Extensive Structured Availability Modeling Environment, in: Proc. 2nd International Conference on the Quantitative Evaluation of Systems (QEST) 2005, IEEE Computer Society Press, 2005, pp. 253–254.

[19] R. Sahner, K. Trivedi, A. Puliafito, Performance and Reliability Analysis of Computer Systems, Kluwer Academic Publishers, 1996.

[20] S.-K. Kuo, S.-K. Lu, F.-M. Yeh, Determining Terminal-Pair Reliability Based On Edge Expansion Diagrams Using OBDD, IEEE Transactions on Reliability 48 (1999) 234–246.

[21] Y. Saad, M. Schultz, GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Stat. Comput. 7 (3) (1986) 856–869.

[22] K. Lampka, M. Siegle, Activity-local Symbolic State Graph Generation for High-Level Stochastic Models, in: Proc. of 13th GI/ITG Conference on "Measuring, Modelling and Evaluation of Computer and Communication Systems" (MMB), Nürnberg, Germany, 2006, pp. 245–263.

[23] K. Lampka, M. Siegle, Analysis of Markov Reward Models unsing Zero-suppressed Multi-terminal BDDs, in: 1st. Int. Conf. on Performance Evaluation Methodologies and Tools (Valuetools), Pisa, Italy, ACM press, ISBN 1-59593-504-5 (CD edition), 10 pages, 2006.

[24] S. Minato, Zero-suppressed BDDs for set manipulation in combinatorial problems, in: 30th ACM/IEEE Design Automation Conference, 1993, pp. 272–277.

[25] H. Hermanns, M. Kwiatkowska, G. Norman, D. Parker, M. Siegle, On the use of MTBDDs for performability analysis and verification of stochastic systems, Journal of Logic and Algebraic Programming 56 (1-2) (2003) 23–67.

[26] M. Siegle, K. Lampka, M. Walter, An easy-to-use, efficient tool-chain to analyze the availability of telecommunication equipment, in: Proc. of the 11th International Workshop on Formal Methods for Industrial Critical Systems (FMICS 2006), Springer, LNCS, 2006, pp. 245–264.

[27] D. Parker, Implementation of symbolic model checking for probabilistic systems, Ph.D. thesis, School of Computer Science, Faculty of Science, University of Birmingham (2002).

[28] A. Bobbio, A. Puliafito, M. Scarpa, M. Telek, WEBSpn: A WEB-accessible Petri Net Tool., in: Proc. of WEB-based Modeling & Simulation, San Diego, California, 1998, pp. 137–142.

[29] S. W. Hunter, W. E. Smith, Availability modeling and analysis of a two node cluster, in: Proc. 5th Int. Conf. on Information Systems, Analysis and Synthesis, 1999.

[30] G. Held, Ethernet Networks, Wiley and Sons, 1994.

[31] PICMG Subcommittee 3.0, Advanced Telecom Computing Architecture Specification, PCI Industrial Computers Manufacturers Group, http://www.picmg.org/ (2003).

[32] Departement of Defense, USA, Reliability Prediction of Electronic Equipment, MilHdbk217F Notice 2 (1995).

[33] Siemens AG, Reliability and Quality Specification Failure Rates of Components, Technical Report SN29500 (1986).

[34] D. Tang, M. Hecht, J. Miller, J. Handal, MEADEP: A dependability evaluation tool for engineers, IEEE Transaction on Computers 47 (4) (1998) 443–450.

[35] A. Bondavalli, M. Dal Cin, D. Latella, I. Majzik, A. Pataricza, G. Savoia, Dependability Analysis in the Early Phases of UML Based System Design., Journal of Computer Systems Science and Engineering 16 (2001) 265–275.

[36] I. Majzik, A. Pataricza, A. Bondavalli, Stochastic Dependability Analysis of System Architecture Based on UML Models, Lecture Notes in Computer Science 2677 (2003) 219–244.

[37] J. Dugan, B. Venkataraman, R. Gulati, DIFtree: a software package for the analysis of dynamic fault tree models, in: Proc. of the Reliability and Maintainability Symposium (RAMS), 1997, pp. 64–70.

[38] J. Dugan, K. Sullivan, D. Coppit, Developing a low-cost high-quality software tool for dynamic fault-tree analysis, IEEE Transaction on Reliability 49-59 (1) (2000) 49ff.

[39] D. Coppit, K. J. Sullivan, Galileo: A tool built from Mass-Market Applications, in: Proceedings of the 22nd International Conference on Software Engineering, IEEE Computer Society Press, 2000, pp. 750–753.

[40] M. Bouissou, J. L. Bon, A new formalism that combines advantages of fault-trees and Markov models: Boolean logic Driven Markov Processes, Reliability Engineering and System Safety (2003) 149–163.

[41] S. Distefano, L. Xing, A New Approach to Modeling the System Reliability: Dynamic Reliability Block Diagrams, in: Proc. of the Annual Realiability and Maintainability Symposium (RAMS 2006), IEEE, 2006, pp. 189–195.

[42] S. Distefano, A. Puliafito, System modelling with dynamic reliability block diagrams, in: Proc. of the 2006 European Safety and Reliability Conference (ESREL 2006), Vol. 1, Taylor and Francis, 2006, pp. 141–150.

[43] S. Distefano, M. Scarpa, A. Puliafito, Modeling Distributed Computing System Reliability with DRBD, in: Proc. of the 25th Symposium on Reliable Distributed Systems (SRDS 2006), IEEE, 2006, pp. 106 – 118.

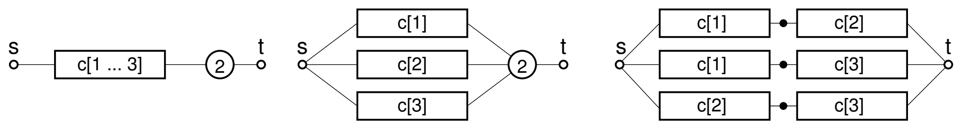

Fig. 1. Usage of k-out-of-N:G edges and arrays. All three diagrams model the same system.

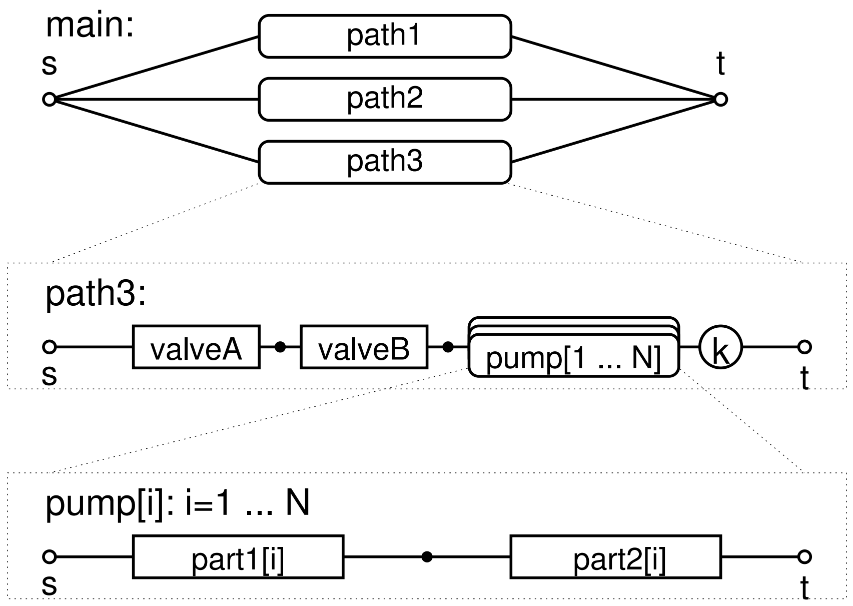

Fig. 2. Hierarchic block diagram. The top-level diagram *main* refers to three sub-diagrams (of which only one is shown). Sub-diagrams may refer to further sub-diagrams. Several identical diagrams can be combined using arrays of diagrams.

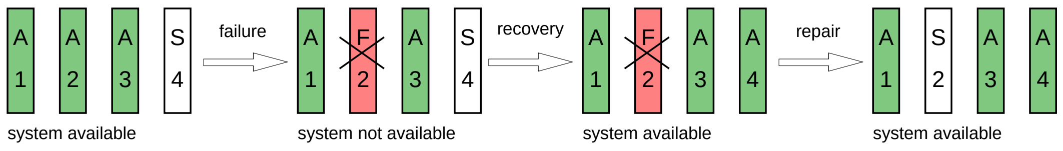

Fig. 3. Four components forming a standby-redundant system. After a component failure, the system is unavailable until the recovery is completed.

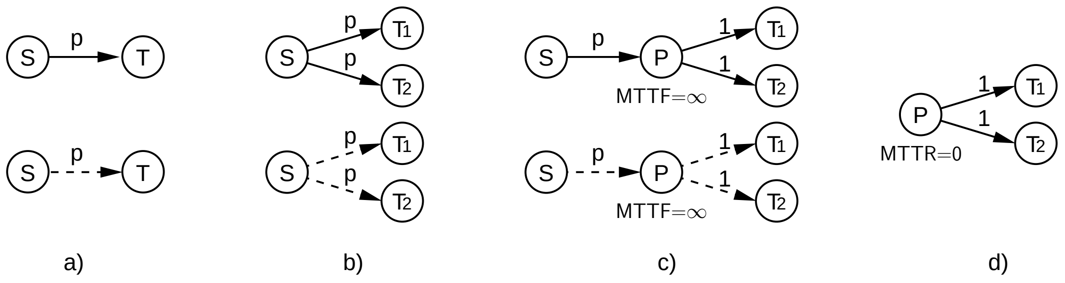

Fig. 4. Failure dependency diagrams (FDD). Destructive propagation is depicted by solid arrows, whereas dashed arrows are used for blocking propagation.

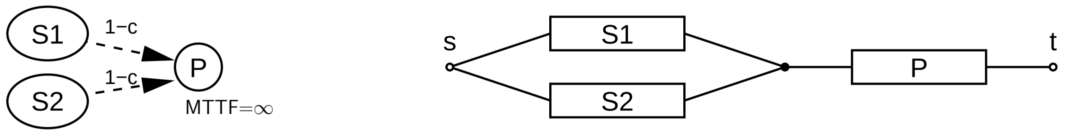

Fig. 5. Imperfect coverage: failure dependency diagram (left) and block diagram (right).

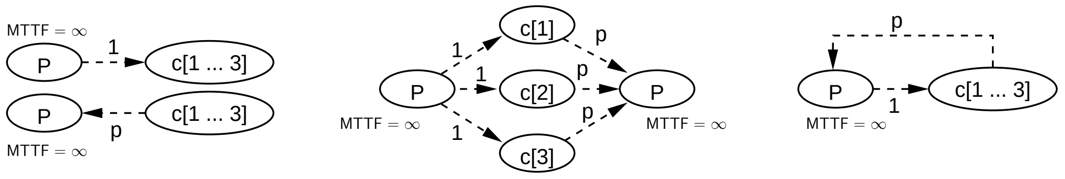

Fig. 6. FDD Arrays. All three diagrams are semantically equal.

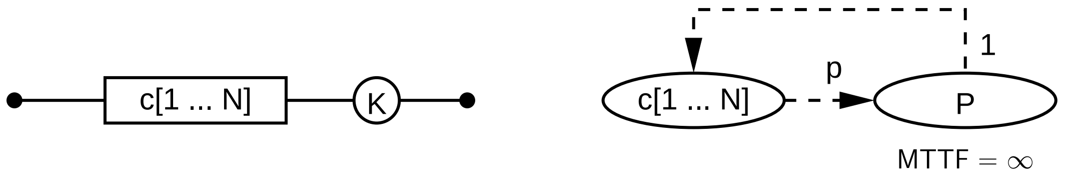

Fig. 7. Generic model of a k-out-of-N:G system with blocking failure propagation. For the analysis, parameters like K, N, and p are substituted by constants.

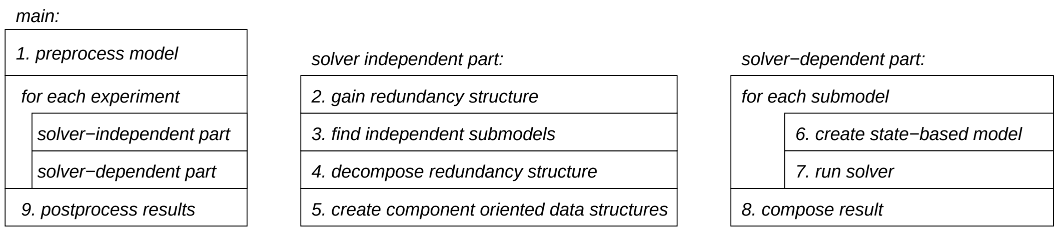

Fig. 8. Overview on the transformation process.

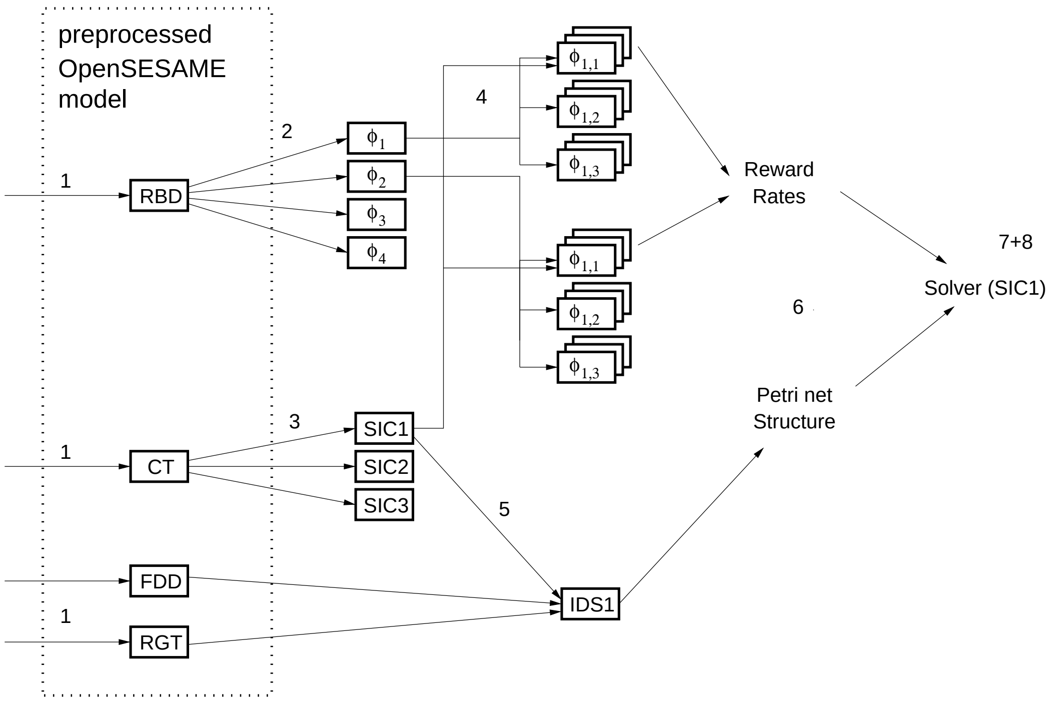

Fig. 9. Data flow diagram of the transformation process. The numbers correspond to the solution steps as defined in Fig. 8.

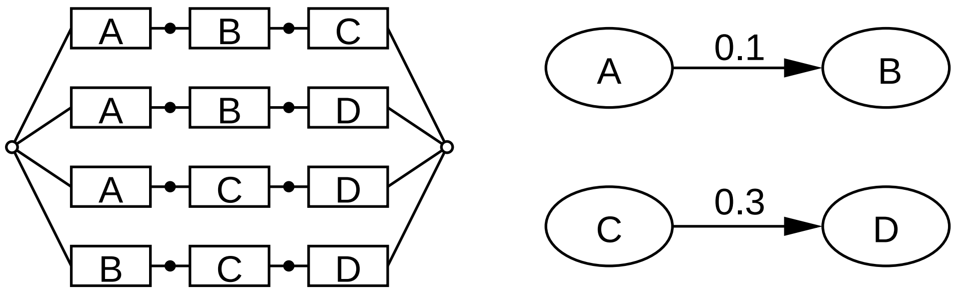

Fig. 10. A model with two sets of inter-dependent components (SIC): $\{A, B\}$ and $\{B, C\}$

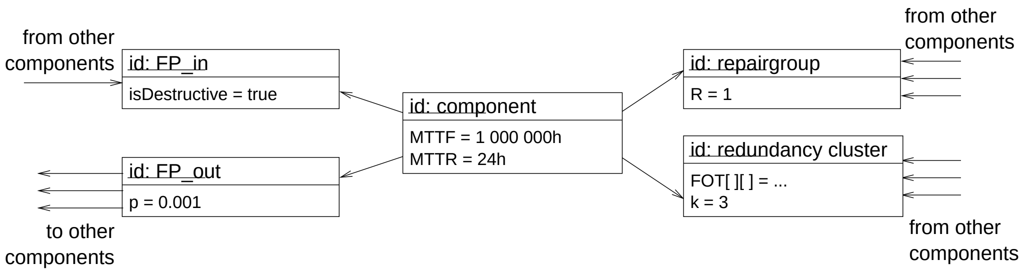

Fig. 11. UML object diagram of the component-oriented data structures.

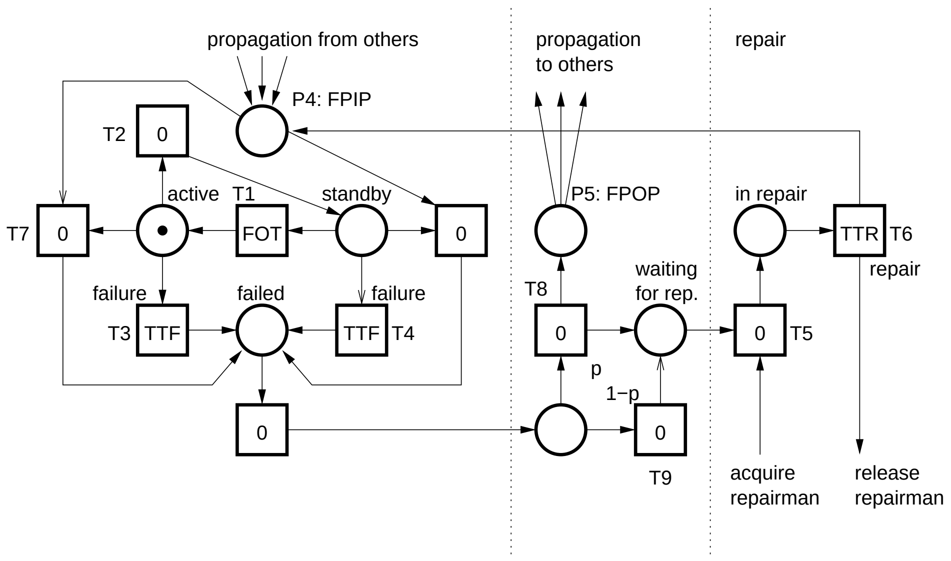

Fig. 12. Petri net of a component.

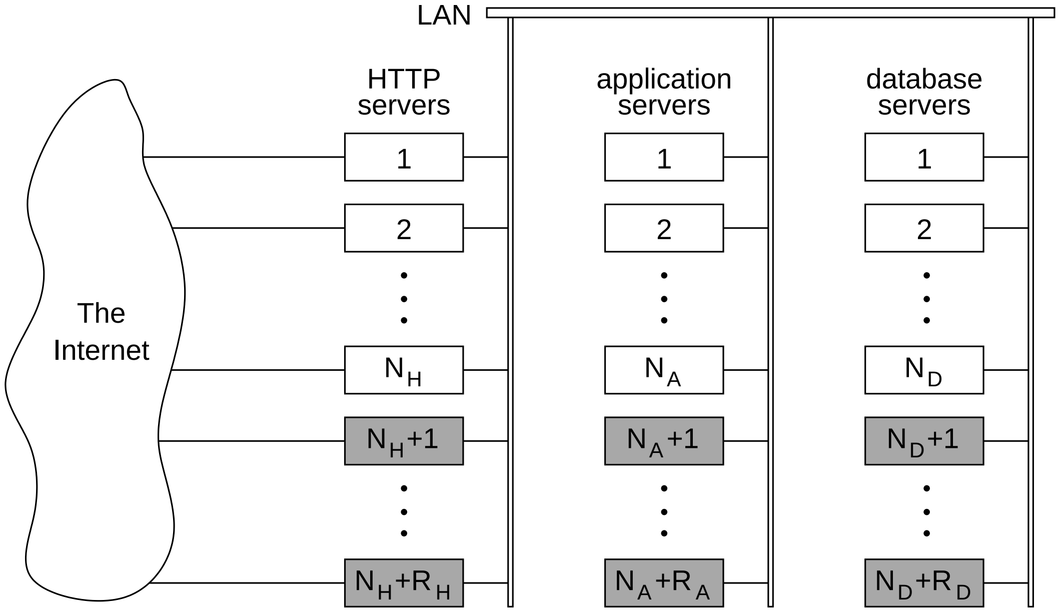

Fig. 13. Structure of a typical distributed web server following a 3-tier approach. It comprises HTTP-, application- and database-servers which are connected via a local area network.

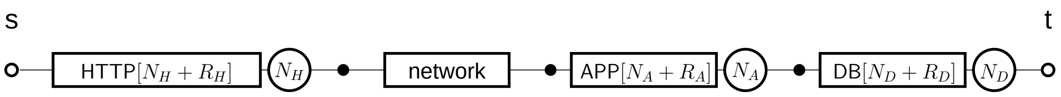

Fig. 14. Reliability block diagram (RBD) of the distributed web server.

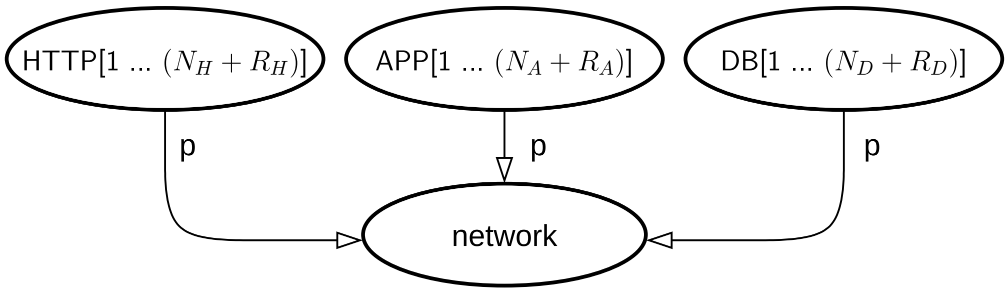

Fig. 15. Failure Dependency Diagram (FDD) of the web server

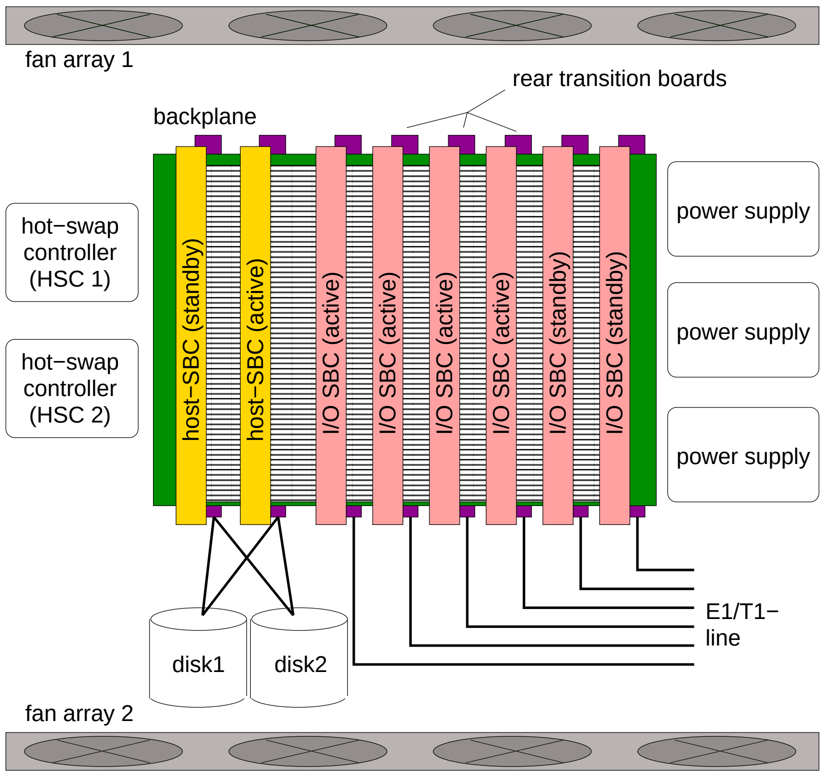

Fig. 16. Schematic drawing of the adjunct processor (AP)

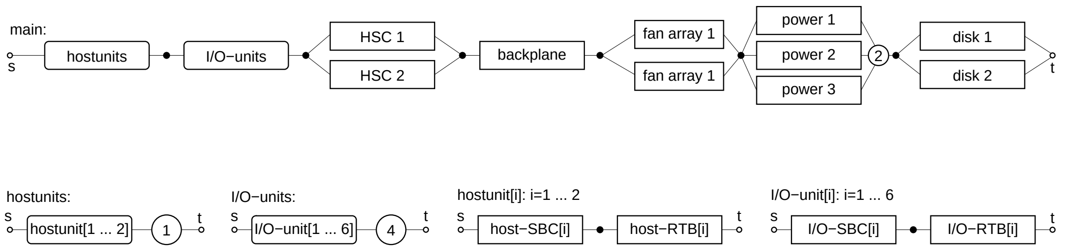

Fig. 17. Hierarchic set of reliability block diagrams modeling the redundancy structure of the adjunct processor.

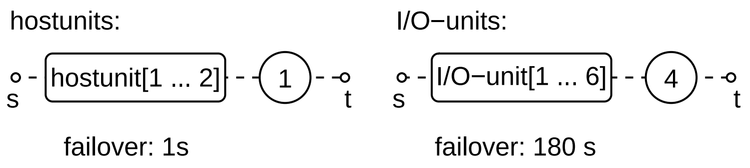

Fig. 18. Modified reliability block diagrams of the adjunct processor including non-zero fail-over times.

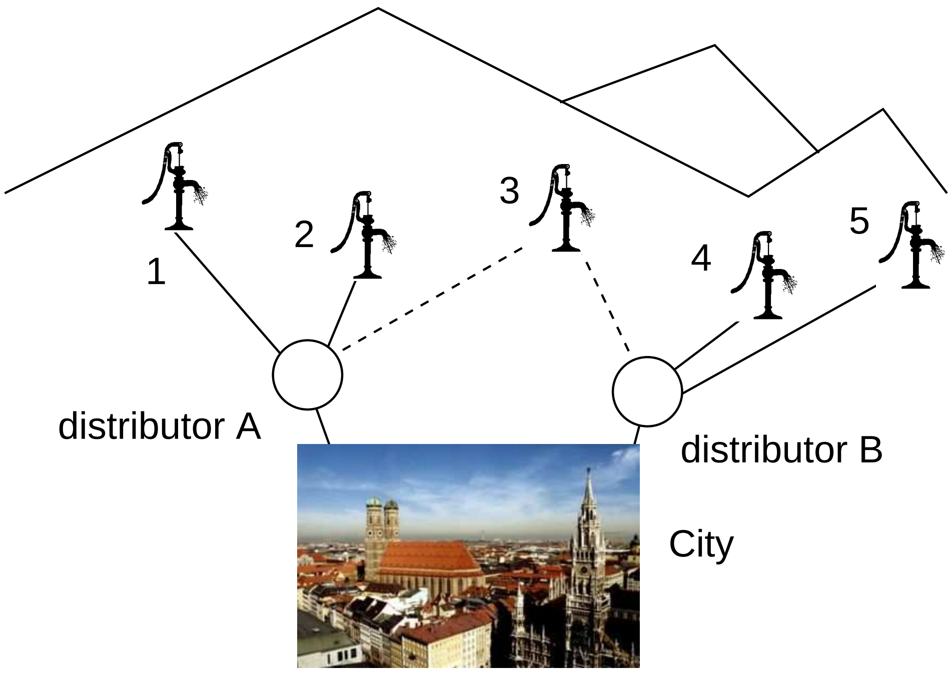

Fig. 19. Water supply system of a city comprising up to five pumps and two water distributors.

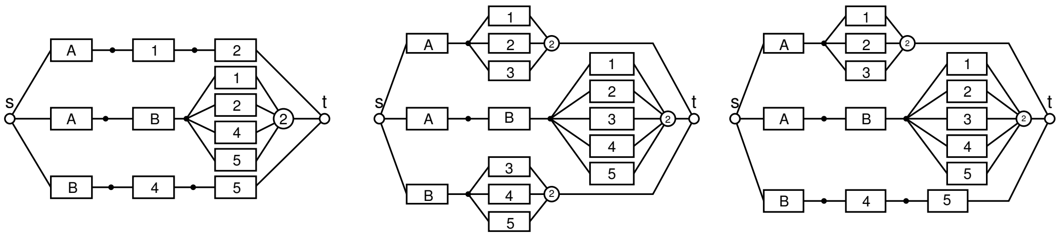

Fig. 20. Reliability block diagram for configuration C1 (left), C2 (middle), and C3 (right).

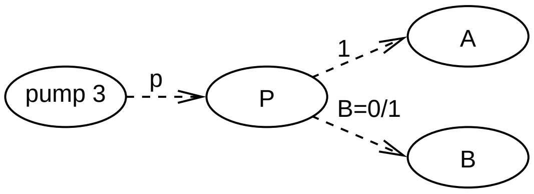

Fig. 21. Failure dependency diagram. If pump 3 fails, with probability p the failure propagation is either propagated to distributor A (B=0) or distributor A and B (B=1).

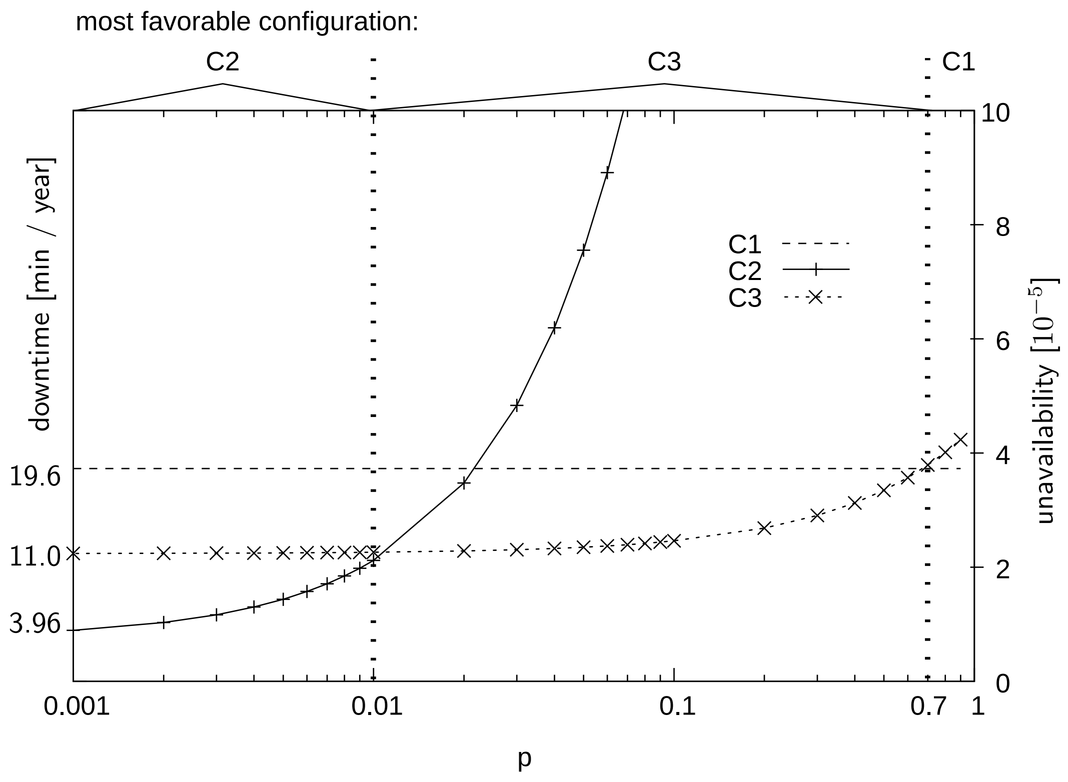

Fig. 22. Unavailability and down-time of the water supply system vs. the failure propagation probability of pump 3 (p). For configuration 1, U has the constant value $3.73 \cdot 10^{-5}$. This configuration is the most favorable if $p > 0.7$. If $p < 0.01$, the best result is obtained using configuration 2. For all other cases ($0.01 < p < 0.7$) configuration 3 should be used.

Table 1

Exemplary component table with four different component types.

component	**number**	**MTTF**	**MTTR**
part1	1	1 000 000 h	24 h
part2	1	2 000 000 h	24 h
valveA	N	1 500 000 h	24 h
valveB	N	1 500 000 h	24 h

Table 2

Components of the distributed web server.

component	**number**	**MTTF**	**MTTR**
HTTP	$N_H + R_H$	168h	0.5h
APP	$N_A + R_A$	168h	0.5h
DB	$N_D + R_D$	168h	0.5h
network	1	1 000 000h	48.0h

Table 3

Results for different degrees of redundancy. Estimated down-time costs are 8000 € per hour.

$R_{H,A,D}$	U	A = 1 - U	annual down-time ($U \cdot 1$ year)	annual costs ($U \cdot 1$ year $\cdot$ 8000 Euro $\cdot h^{-1}$)
0	$1.77 \cdot 10^{-2}$	0.9823	155 h	1 240 000 €
1	$1.27 \cdot 10^{-4}$	0.999873	1.11 h	8 880 €
2	$4.83 \cdot 10^{-5}$	0.9999517	0.423 h	3 384 €
3	$4.80 \cdot 10^{-5}$	0.9999520	0.420 h	3 360 €

Table 4

The unavailability (in ‰) for different babbling node probabilities.

probability	**0**	**0.002**	**0.004**	**0.006**	**0.008**	**0.010**
$R_{H,A,D} = 1$	0.127	0.180	0.233	0.286	0.339	0.392
$R_{H,A,D} = 2$	0.0483	0.120	0.191	0.262	0.333	0.404

Table 5

Components of the adjunct processor

Component	**number**	**MTTF**	**MTTR**
host-SBC	2	100 000 h	5 min
host-RTB	2	1 100 000 h	30 min
I/O-SBC	6	200 000 h	5 min
I/O-RTB	6	1 250 000 h	30 min
disk	2	1 000 000 h	30 min
HSC	2	100 000 h	5 min
backplane	1	5 000 000 h	1 h
fan array	2	17 500 h	5 min
power supply	3	1 450 000 h	5 min